TGNN: GRAPH NEURAL NETWORKS FOR LARGE-SCALE INDUSTRIAL ADVERTISING SCENARIOS

Anonymous authors
Paper under double-blind review

ABSTRACT

Deploying Graph Neural Networks (GNNs) in industrial advertising is fundamentally hindered by three intersecting challenges: extreme structural sparsity in cold-start scenarios, hundred-billion-edge scalability limits, and millisecond real-time serving constraints. We present TGNN, an industrial-scale GNN framework designed to systematically overcome these bottlenecks. To alleviate severe sparsity, where over 34% of graph nodes have a degree below five, TGNN leverages Large Language Models (LLMs) to construct semantically meaningful connections among sparse nodes, combining this with fine-grained feature learning and cross-domain data augmentation to alleviate the sparsity issue. To support massive graphs exceeding 2 billion nodes and 100 billion edges, we design EasyGraph, a distributed graph database for efficient graph storage and sampling, and AngelGraph, a large-scale graph computing platform that optimizes GNN training. Furthermore, we propose a hybrid stream-batch pipeline that guarantees strict temporal consistency and low-latency feature propagation. Online A/B testing demonstrates significant business gains: overall GMV increases by 3.43% (new advertisement 4.80%) in social feeds and by 2.77% (new advertisement 3.74%) in short-video feeds. TGNN provides a complete and highly scalable solution for deploying GNNs in real-world advertising recommendation systems.

1 INTRODUCTION

Online advertising aims to balance user experience, advertisers' Return on Investment (ROI), and platform revenue through precise user interest matching. A major industrial advertising platform's ecosystem, spanning social-feed and short-video scenarios, serves over 1 billion users and involves hundreds of billions of interactions — a massive, heterogeneous network that poses unique challenges for industrial-scale recommendation systems.

The inherent characteristics of advertising — scattered cross-scenario behaviors, acute cold-start issues for new advertisements, and elusive semantic correlations — limit recommendation accuracy and diversity in two ways: (1) *acute data sparsity*, where newly launched ads lack interaction history and 34% of graph nodes have a degree below five (Section 4.4); and (2) *disjointed multi-scenario signals*, where fragmented user actions cannot be integrated into a unified representation without explicit structural modeling.

Graph Neural Networks (GNNs) (Kipf & Welling, 2017; Hamilton et al., 2017; Veličković et al., 2018) offer a powerful paradigm: their message-passing mechanism bridges data-rich neighbors and long-tail items, and their explicit relational modeling unifies fragmented heterogeneous signals. Nevertheless, deploying GNNs at web scale in advertising introduces three fundamental challenges:

- **Extreme Structural Isolation**: Over 34% of graph nodes have a degree below five and new advertisements have no interactions, which neutralizes standard message passing.
- **Hundred-Billion Edge Scalability**: 2 billion nodes and 100 billion edges induce immense memory and communication overhead, and viral "super-nodes" easily overwhelm conventional distributed graph systems (Wan et al., 2022; Gandhi & Iyer, 2021; Zhang et al., 2020).

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

- **Millisecond Latency Paradox**: Multi-hop aggregation conflicts with real-time serving constraints (latency under 20 ms, freshness within the 5-minute production SLA), forcing a trade-off between timeliness and accuracy.

To surmount these challenges, we propose TGNN: it integrates GNNs with Large Language Models (LLMs) to alleviate sparsity via semantic enrichment, is powered by a dedicated distributed graph storage and computing infrastructure, and is deployed through a unified stream-batch pipeline. Deployed across all traffic of a major advertising platform, TGNN is validated by extensive online A/B testing. Our contributions are:

- **Semantic-Enhanced Graph Representation Learning Paradigm**: We integrate GNNs with LLMs to infer multi-dimensional semantic connections for sparse nodes; combined with data augmentation, contrastive learning and cross-scenario feature supplementation, this paradigm adds 1.2 billion semantic edges in production, covering 92% of advertisements and accounting for 4.7% of all advertisement out-edges.
- **Development of a Large-Scale Distributed Graph Computing Infrastructure**: We design EasyGraph for distributed graph storage and AngelGraph for optimized GNN computing; optimizations in distributed architecture, graph partitioning, dynamic task scheduling and subgraph sampling achieve million-level QPS for 2-hop (5×5) subgraph sampling (i.e., millions of sampling calls per second, each request issuing several calls) with a TP-999 latency strictly under 10 ms.
- **Industrial Real-Time Recommendation Solution**: We build a hybrid stream-batch training and inference pipeline ensuring low-latency serving and strict training–inference feature consistency; TGNN processes 200 million incremental training samples and 34 billion inference requests daily (an average of about 3.9×10^5 requests per second), refreshing user embeddings within 2 minutes of a real-time interaction (measured end-to-end; the production freshness SLA is 5 minutes).

1.1 Graph Neural Networks for Advertising

Graph Neural Networks (GNNs) (Kipf & Welling, 2017; Hamilton et al., 2017; Veličković et al., 2018) provide a robust foundation for representation learning by localized message passing; in advertising they have evolved into architectures tailored to heterogeneous and sparse data, e.g., Fi-GNN (Li et al., 2019) and AdsGNN (Li et al., 2021). Sparsity is addressed either by Graph Contrastive Learning (GCL), which augments the topology to extract robust representations from sparse feedback (SimGCL (Yu et al., 2022), DivGCL (Gong et al., 2025)), or by knowledge-enhanced methods injecting side information, most recently LLM-based preference priors (RecMind (Xue et al., 2025), RARE (Liu et al., 2025)). Unlike GCL methods that perturb existing interactions, TGNN *actively* constructs semantically meaningful connections with LLMs and hierarchical CPV tags, so that ads with zero interactions inherit information from mature ones.

1.2 Industrial GNNs Advertising Systems and Infrastructures

In industrial practice, the challenge shifts from theoretical modeling to web-scale scalability and efficiency. PinSage (Ying et al., 2018) pioneered random-walk convolutions at billion-node scale; LiGNN (Borisyuk et al., 2024) adds Personalized-PageRank sampling and temporal modeling; OmniSAGE (Badrinath et al., 2025) generalizes this into a multi-entity framework with Transformer (Vaswani et al., 2017)-style aggregators; TTGL (Zhu et al., 2025) targets universal graph learning with feature consistency across scenarios; GiGL (Zhao et al., 2025) abstracts distributed processing for hundred-billion-edge graphs (Fey & Lenssen, 2019; Wang et al., 2019a). Graph-native databases such as Neo4j (Lal, 2015), TigerGraph (Deutsch et al., 2019) and NebulaGraph (Wu et al., 2022) now offer vector search and real-time graph updates, but they either lack graph sampling or cannot satisfy the scale and performance requirements of industrial advertising. Industrial frameworks such as EmbedX (Zou et al., 2023) and LPS-GNN (Cheng et al., 2025) scale to billion-node graphs, yet EmbedX supports heterogeneity only rudimentarily and LPS-GNN is confined to static homogeneous networks; in contrast, TGNN targets hundred-billion-edge *heterogeneous* graphs with multi-dimensional load balancing and real-time serving (Section 3).

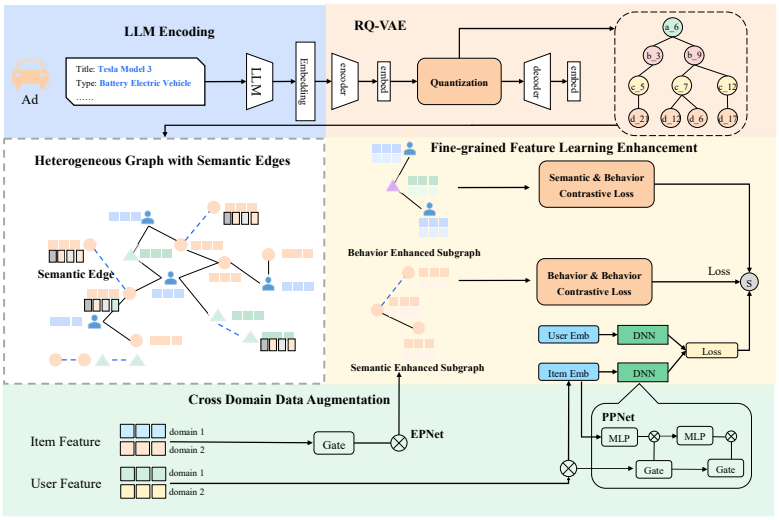

Figure 1: An illustration of TGNN model architecture.

2 GNN Modeling and Training

TGNN addresses severe data sparsity through semantic edge construction, fine-grained feature learning and cross-domain supplementation; this section details its architecture.

As shown in Figure 1, TGNN utilizes a heterogeneous Graph Attention Network (GAT) (Veličković et al., 2018) built on custom metapaths. Subgraph representations are derived through weighted metapath aggregation using either an attention mechanism or PEPNet (Chang et al., 2023)-based gating. To enhance robustness, we implement data augmentation via contrastive learning across subgraphs (e.g., behavior vs. semantic). The subgraph representations are then aggregated into the source- and target-node embeddings. A personalized feature extraction module combines them with user-specific features via gating and an MLP. The final user and item embeddings are optimized using behavioral edges between users and items as supervision labels.

2.1 Problem Formulation and Notations

Let $\mathcal{U}$ and $\mathcal{A}$ denote the sets of users and advertisements ($|\mathcal{A}| > 10^8$). We model the advertising ecosystem as a weighted heterogeneous graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \phi, \psi, w)$, where $\mathcal{V} = \mathcal{U} \cup \mathcal{A} \cup \mathcal{S}$ ($\mathcal{S}$ is the set of semantic attribute nodes such as CPV tags, industries and categories), $\phi : \mathcal{V} \to \mathcal{T}_v$ and $\psi : \mathcal{E} \to \mathcal{T}_e$ are the node-type and edge-type mappings, and $w : \mathcal{E} \to [0, 1]$ assigns a strength to every edge. The edge set partitions into two disjoint components, $\mathcal{E} = \mathcal{E}^{beh} \dot{\cup} \mathcal{E}^{sem}$: the *behavior* subgraph $\mathcal{G}^{beh}$ induced by logged user–ad interactions, and the *semantic* subgraph $\mathcal{G}^{sem}$ induced by LLM-derived and attribute-derived associations. Because $\mathcal{E}^{sem}$ is *not* a function of user behavior, it is available at launch time — the property exploited throughout Section 2.3.

Encoder. Given input features $\mathbf{x}_v$, the heterogeneous GAT encoder f_θ performs L layers of edge-weighted attention. At layer l, with $\mathbf{h}_v^{(0)} = \mathbf{x}_v$,

$$\alpha_{vu}^{(l)} = \frac{\exp\big(\mathbf{a}^\top [\mathbf{W}^{(l)}\mathbf{h}_v^{(l)} \| \mathbf{W}^{(l)}\mathbf{h}_u^{(l)}] + \log w_{vu}\big)}{Z_v^{(l)}}, \tag{1}$$

$$Z_v^{(l)} = \sum_{k \in \mathcal{N}(v)} \exp\big(\mathbf{a}^\top [\mathbf{W}^{(l)}\mathbf{h}_v^{(l)} \| \mathbf{W}^{(l)}\mathbf{h}_k^{(l)}] + \log w_{vk}\big),$$

$$\mathbf{h}_v^{(l+1)} = \sigma\Big(\mathbf{W}_s^{(l)}\mathbf{h}_v^{(l)} + \textstyle\sum_{u \in \mathcal{N}(v)} \alpha_{vu}^{(l)} \mathbf{W}^{(l)}\mathbf{h}_u^{(l)}\Big), \tag{2}$$

Here $\|$ is concatenation, $\mathbf{W}_s^{(l)}$ is the self-loop transformation, and $\log w_{vu}$ enters the softmax as a logit bias, so that weak (noisy) edges are down-weighted; restricting Eq. 2 to $\mathcal{E}^{beh}$ and $\mathcal{E}^{sem}$ yields the view-specific embeddings $\mathbf{e}_v^{beh}, \mathbf{e}_v^{sem} \in \mathbb{R}^d$ fused in Section 2.5.

Objective and evaluation. Given the interaction history of user u up to day T, a candidate ad a is retrieved when its score $s(u,a) = \cos(\mathbf{e}_u, \mathbf{e}_a)$ is among the Top-K of $\mathcal{A}$. Training minimizes

$$\mathcal{L}(\theta) = \mathcal{L}_{main} + \lambda_{ssl}\mathcal{L}_{ssl} + \lambda_{reg}\|\theta\|_2^2, \tag{3}$$

$$\mathcal{L}_{main} = -\mathbb{E}_{(u,a)} \log \frac{\exp\big(s(u,a)/\tau\big)}{\exp\big(s(u,a)/\tau\big) + \sum_{j=1}^{N^-} \exp\big(s(u,a_j^-)/\tau\big)}, \tag{4}$$

where $\mathcal{L}_{main}$ is a sampled-softmax loss over behavioral positives (u,a) with N^- sampled negatives, and $\mathcal{L}_{ssl}$ is the hierarchical contrastive term of Section 2.4. Algorithm 2 in Appendix H summarizes one complete training step.

2.2 Graph Construction

The three node types of Section 2.1 are instantiated as *user nodes* (profile, behavioral-statistical and scenario-preference features), *advertisement nodes* (attributes, statistics and semantic features), and *semantic nodes* (categories, industries and CPV tags). The two edge families use different sampling windows to control the graph scale: *behavioral edges* are split into click-stream edges over a 7-day window and long-term interest edges over a 90-day window, while *semantic edges* follow Section 2.3. The resulting graph has over 2×10^9 nodes and 10^{11} edges. In multi-day logs the same pair may carry several edges of the same type, aggregated into the single strength w_{vu} of Eq. 1, which removes sampling redundancy.

2.3 Constructing Semantic Edges

A freshly launched ad has no behavioral edge, so Eq. 2 degenerates to its self-loop term. We therefore add semantic edges independent of user behavior, in two forms: (i) Item-to-Item (I2I) edges from LLM-based *semantic IDs*, letting a new ad receive messages from similar mature ads; (ii) I2CPV edges connecting users and ads to auxiliary nodes of a hierarchical Category-Property-Value (CPV) tagging system.

2.3.1 Constructing I2I Edges with LLM

We generate semantic IDs by combining an LLM, RQ-VAE (Lee et al., 2022) and a trie tree: the LLM maps multi-source ad data (text, category, description) to a semantic vector $\mathbf{s}_a = \mathrm{LLM}(x_a) \in \mathbb{R}^{d_{RQ}}$, and RQ-VAE quantizes it with M layers of residual quantization, where layer l maintains a codebook $C^l = \{\mathbf{e}_k^l\}_{k=1}^K$:

$$\mathbf{z}_a = \sum_{l=1}^{M} \mathbf{e}^l_{c_a^l}, \quad c_a^l = \arg\min_k \|\mathbf{r}_a^l - \mathbf{e}_k^l\|_2^2, \quad \mathbf{r}_a^{l+1} = \mathbf{r}_a^l - \mathbf{e}^l_{c_a^l}, \quad \mathbf{r}_a^1 = \mathbf{s}_a, \tag{5}$$

where $\mathbf{r}_a^l$ is the l-th residual and c_a^l the index of ad a in codebook l; the *semantic ID* $\mathrm{sid}(a) = (c_a^1, \ldots, c_a^M) \in [K]^M$ represents K^M distinct concepts with only KM stored vectors, and the codebooks use the straight-through estimator.

Trie-based I2I construction. A hierarchical trie is built over $\mathrm{sid}(\cdot)$: the root is the generic category and the node at depth l is the codeword c^l, so the *common-prefix depth* of two ads is $\ell(a,b) = \max\{l \in \{0,\ldots,M\} : c_a^{1:l} = c_b^{1:l}\}$. The I2I weight is the normalized prefix overlap, truncated at a minimum depth $\ell_{\min}$ so that only genuinely similar ads are linked:

$$w_{ab}^{i2i} = \frac{\ell(a,b)}{M} \cdot \mathbb{1}\big[\ell(a,b) \geq \ell_{\min}\big], \tag{6}$$

and each ad keeps only its Top-D heaviest I2I edges ($D \ll |\mathcal{A}|$), so that the semantic degree stays bounded and message passing remains $O(D)$ per node.

Proposition 2.1 (Quality of trie-retrieved semantic neighbors)**.** *Assume the RQ-VAE residuals decay geometrically, $\|\mathbf{r}_a^l\|_2 \le \rho^l \|\mathbf{s}_a\|_2$ for some $\rho \in (0,1)$ (an empirical property of the trained quantizer, invoked only to make the depth–similarity trade-off explicit). Then for any two ads with $\ell(a,b) = \ell$ and $\|\mathbf{s}_a\|_2 = \|\mathbf{s}_b\|_2 = \|\mathbf{s}\|_2$,*

$$\cos(\mathbf{s}_a, \mathbf{s}_b) \;\ge\; 1 - 2\Big((1+\rho)\frac{\rho^{\ell+1}}{1-\rho} + \rho^M\Big)^2. \tag{7}$$

Proposition 2.1 (proof in Appendix H) certifies Eq. 6: similarity grows exponentially with common-prefix depth, so ℓ/M is a monotone proxy of semantic similarity, and any target $1-\kappa$ is met by a constant depth $\ell_{\min} = O(\log_\rho \sqrt{\kappa})$, independent of $|\mathcal{A}|$; trie retrieval costs $O(M)$ per ad versus $O(|\mathcal{A}| \log |\mathcal{A}|)$ for exhaustive cosine search (Algorithm 1, Appendix C.2).

2.3.2 New Item Generalization Based on Hierarchical Tagging System

The Category-Property-Value (CPV) system assigns structured attribute tags to users and ads, providing informative gain for ad modeling and refining I2I construction. Let $\mathcal{T}_{CPV}(a) \subseteq \mathcal{T}_{CPV}$ be the tag set of ad a, n_t the number of ads carrying tag t, and $\mathrm{lvl}(t) \in \{\text{category}, \text{property}, \text{value}\}$ its hierarchy level. Generic tags (e.g., a broad industry label) carry little discriminative signal, so we weight tag edges by an inverse-document-frequency factor and by hierarchy level, and aggregate user–tag edges with a recency kernel:

$$w_{a,t}^{cpv} = \underbrace{\log \frac{|\mathcal{A}|}{1+n_t}}_{\mathrm{IDF}(t)} \cdot \kappa_{\mathrm{lvl}(t)}, \qquad w_{u,t}^{cpv} = \frac{\sum_{a \in \mathcal{A}_u^+} w_{a,t}^{cpv} \, \exp(-\Delta t_{ua}/\tau_d)}{\sum_{a \in \mathcal{A}_u^+} \exp(-\Delta t_{ua}/\tau_d)}, \tag{8}$$

where $\kappa_{\text{cat}} > \kappa_{\text{prop}} > \kappa_{\text{val}}$, $\mathcal{A}_u^+$ is the set of ads interacted by u, and Δt_{ua} is the age of the interaction. The number of tag edges is $O(\sum_a |\mathcal{T}_{CPV}(a)|)$, linear in the tag budget rather than quadratic in $|\mathcal{A}|$; these edges let a new ad anchor to potential users through attribute associations.

2.4 Fine-grained Feature Learning Enhancement

Fine-grained features such as ID embeddings require large volumes of data to be learned well; under the inherent sparsity of advertising logs their convergence is suboptimal, so we apply noise-based data augmentation and hierarchical contrastive learning, which amplify the contribution of long-tail samples during training.

View construction. For each node i we draw two independent perturbations to the encoder input:

$$\Delta_i^{(k)} = \varepsilon \cdot \frac{\bar{\Delta}_i^{(k)}}{\|\bar{\Delta}_i^{(k)}\|} \odot \mathrm{sign}(\mathbf{x}_i), \quad \bar{\Delta}_i^{(k)} \sim \mathcal{U}(0,1)^d, \quad k \in \{1,2\}, \tag{9}$$

so that $\Delta_i^{(k)}$ lies on the hypersphere of radius ε; the augmented views $\mathbf{x}_i^{(k)} = \mathbf{x}_i + \Delta_i^{(k)}$ are encoded by f_θ restricted to each subgraph (Section 2.1).

Hierarchical contrastive objective. Let $\mathcal{B}$ be a mini-batch of N nodes and $\mathbf{h}_i^{(k)} = f_\theta(\mathbf{x}_i + \Delta_i^{(k)})$ the two views of node i. All three strategies below instantiate the same InfoNCE objective (Oord et al., 2018) with different pairing rules:

$$\mathcal{L}_{NCE}\big(\{\mathbf{u}_i\}, \{\mathbf{v}_i\}\big) = -\frac{1}{N}\sum_{i=1}^{N} \log \frac{\exp\big(\mathrm{sim}(\mathbf{u}_i, \mathbf{v}_i)/\tau\big)}{\sum_{j=1}^{N} \exp\big(\mathrm{sim}(\mathbf{u}_i, \mathbf{v}_j)/\tau\big)}, \quad \mathrm{sim}(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u}^\top \mathbf{v}}{\|\mathbf{u}\|\|\mathbf{v}\|}. \tag{10}$$

- *Intra-subgraph contrast*: the two views of the same node inside subgraph $g \in \{beh, sem\}$ form the positive pair, $\mathcal{L}_{intra} = \sum_g \mathcal{L}_{NCE}\big(\{\mathbf{h}_i^{g,(1)}\}, \{\mathbf{h}_i^{g,(2)}\}\big)$, strengthening long-tail nodes and robustness to noisy interactions.
- *Cross-subgraph contrast*: the behavior view and the semantic view of the *same* ad form the positive pair, $\mathcal{L}_{cross} = \mathcal{L}_{NCE}\big(\{\mathbf{e}_a^{beh}\}_{a\in\mathcal{B}}, \{\mathbf{e}_a^{sem}\}_{a\in\mathcal{B}}\big)$, explicitly transferring behavior signal into the semantic branch.

- *Top-level embedding contrast*: contrast is applied to pooled subgraph summaries $\mathbf{z}_i^{(k)} = \mathrm{Pool}\big(\{\mathbf{h}_j^{(k)} : j \in \mathcal{S}_i\}\big)$, $\mathcal{L}_{top} = \mathcal{L}_{NCE}\big(\{\mathbf{z}_i^{(1)}\}, \{\mathbf{z}_i^{(2)}\}\big)$, aligning high-order neighborhood structure rather than individual nodes.

The three terms enter Eq. 3 as $\mathcal{L}_{ssl} = \mathcal{L}_{intra} + \mu_1 \mathcal{L}_{cross} + \mu_2 \mathcal{L}_{top}$.

Why noise contrast amplifies long-tail samples. Writing $p_{ij} = \mathrm{softmax}_j\big(\mathrm{sim}(\mathbf{u}_i, \mathbf{v}_j)/\tau\big)$, the gradient of Eq. 10 w.r.t. an anchor $\mathbf{u}_i$ is

$$\frac{\partial \mathcal{L}_{NCE}}{\partial \mathbf{u}_i} = -\frac{1}{\tau}\Big((1 - p_{ii})\,\mathbf{v}_i - \sum_{j \neq i} p_{ij}\,\mathbf{v}_j\Big). \tag{11}$$

The attraction towards the positive view is scaled by $1 - p_{ii}$: well-aligned nodes ($p_{ii} \to 1$, typically popular ads) receive a vanishing gradient, whereas poorly aligned nodes ($p_{ii} \to 1/N$, long-tail and cold-start ads) receive a nearly full-magnitude gradient. Noise contrast thus acts as an adaptive gradient re-weighting over the degree distribution — exactly the amplification required by sparse advertising data.

2.5 Cross-Domain Data Augmentation

To avoid exacerbating sparsity with independent per-domain models, we train a single unified GNN, using diverse traffic scenarios to bolster the representations of new advertisements.

User side. For personalization we adopt PEPNet (Chang et al., 2023), where a Gate Neural Unit (Gate NU) processes domain-related features $E(df) \in \mathbb{R}^k$ and modulates all other features $E(sf) \in \mathbb{R}^k$:

$$\delta_{domain} = \mho_{ep}\big(E(df) \oplus \oslash(E(sf))\big), \qquad O_{ep} = \delta_{domain} \odot E(sf), \tag{12}$$

where $\oplus$ denotes concatenation, $\oslash$ denotes stop gradient, and δ_{domain} is a domain-specific personalized modulation that resolves semantic inconsistencies across heterogeneous feature spaces. We construct scenario-specific user subgraphs partitioned by traffic domain and use the $site_id$ of each domain as the gate input; per-domain subgraph embeddings are fused by the resulting weights:

$$\pi_d(u) = \frac{\exp\big(\mathbf{w}_d^\top[\mathbf{s}_d \oplus \oslash(\mathbf{e}_u^{(d)})]\big)}{\sum_{d' \in \mathcal{D}} \exp\big(\mathbf{w}_{d'}^\top[\mathbf{s}_{d'} \oplus \oslash(\mathbf{e}_u^{(d')})]\big)}, \qquad \mathbf{e}_u = \sum_{d \in \mathcal{D}} \pi_d(u)\, O_{ep}^{(d)}, \tag{13}$$

where $\mathbf{s}_d$ is the scenario embedding and $\mathcal{D}$ is the set of traffic domains.

Advertisement side. Scenario partitioning applies only to behavioral subgraphs: semantic subgraphs (I2I and I2CPV edges) remain shared across domains, since ad attributes are scenario-invariant. A gating network driven by ad-maturity features therefore treats behavioral subgraphs as "in-domain experts" and semantic subgraphs as "shared experts". With $\mathbf{m}_a = [\mathrm{age}_a, \log(1 + n_a^{clk}), \log(1 + n_a^{imp})]$,

$$\big[g_a^{beh}, g_a^{sem}\big] = \mathrm{softmax}\Big(\mathbf{W}_2\,\mathrm{ReLU}\big(\mathbf{W}_1[\mathbf{m}_a \oplus \oslash(\mathbf{e}_a^{beh}) \oplus \oslash(\mathbf{e}_a^{sem})]\big)\Big), \tag{14}$$

$$\mathbf{e}_a = g_a^{beh} \odot \mathbf{e}_a^{beh} + g_a^{sem} \odot \mathbf{e}_a^{sem}. \tag{15}$$

Since the gate is a function of $\mathbf{m}_a$, a freshly launched ad ($n_a^{clk} \approx 0$) defaults to the shared semantic expert available at launch time, whereas a mature ad progressively shifts mass to the domain-specific behavior expert: a continuous, data-driven interpolation between the cold-start and warm regimes, which also enables collaborative modeling across scenarios. Section 3 describes how these components are executed at scale; Algorithm 2 (Appendix H) assembles them into one training step.

3 System Design for Scalability and Freshness

3.1 Overall Architecture Design

Deploying GNNs at this scale imposes constraints beyond algorithmic design: latency (<20 ms), data-freshness (production SLA: 5 min; measured near-line refresh below 2 min, Table 2) and offline–online

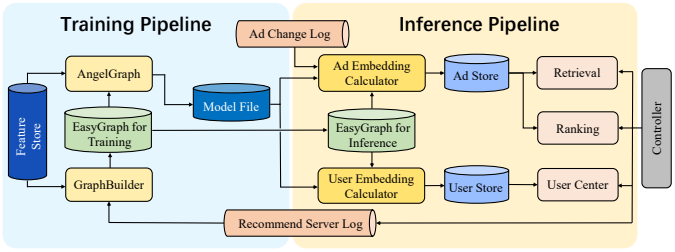

Figure 2: An illustration of the overall TGNN architecture.

consistency (Section 1). We therefore implement the hybrid stream-batch near-line computation architecture of Figure 2, which balances serving performance against data freshness. The *User Embedding Calculator* subscribes to the *Recommend Server Log* so that every user request triggers a user-side streaming computation; symmetrically, the *Item Embedding Calculator* subscribes to the *Ad Change Log* and recomputes an ad embedding whenever its metadata or status changes. A daily batch process re-calibrates all active users and advertisements against the latest model parameters without interrupting serving, and timestamp-based versioning prevents stale batch updates from overwriting newer streaming data.

Storage, computation and features. EasyGraph uses a single-writer multi-reader design that stores outgoing edges in contiguous adjacency lists and partitions dense super-nodes with an attribute-indexed B-Tree; together with request coalescing, graph caching and time-windowed MVCC (which bounds sampling by historical timestamps and eliminates temporal leakage), it achieves a TP-999 latency below 10 ms for standard 2-hop (5×5) sampling. AngelGraph adopts a hybrid Parameter Server (Li et al., 2013) + Spark (Zaharia et al., 2010) architecture (Jiang et al., 2020): it samples subgraphs from EasyGraph inside Spark executors, feeds batches to distributed PyTorch (Paszke et al., 2019), natively supports custom meta-paths and multi-subgraph contrastive learning, and resolves straggler workers through load-aware super-node partitioning and pull-based dynamic task scheduling. Features are materialized in EasyGraph by *GraphBuilder*, an event-driven component that fetches raw attributes from a centralized *Feature Store*, runs the extraction pipeline, and writes the processed features back to EasyGraph, guaranteeing feature freshness and train–serve consistency. Appendices D–E detail the memory layouts, partitioning and scheduling algorithms, and serialization benchmarks.

4 Experiments

We evaluate TGNN through large-scale offline experiments and online A/B testing on an industrial advertising platform, covering two scenarios with distinct content ecosystems: **Dataset A**, a social-media feed with heterogeneous content types and dense interactions, and **Dataset B**, a high-throughput short-video platform with rapid consumption cycles and real-time interest shifts.

4.1 Offline Results Comparison

Protocol and baselines. We adopt a next-day prediction protocol: embeddings are generated on day T, and an interaction on day $T + 1$ is a "hit" if the ground-truth ad lies within the Top-K most similar candidates; we report all ten metrics (AUC, Recall@K, Precision@K and F1@K for $K \in \{1, 20, 100\}$). We compare against **HiGNN** (our production baseline on a two-hop behavior graph), three external baselines — **SimGCL** (Yu et al., 2022), **GraphSAGE** (Hamilton et al., 2017) and **HAN** (Wang et al., 2019b) (the heterogeneous graph with all TGNN-specific components removed) — and three ablations, each removing exactly one TGNN-specific component: **TGNN-w/o-SE** (no semantic edges, §2.3), **TGNN-w/o-HC** (no hierarchical contrastive learning, §2.4) and **TGNN-w/o-PG** (no PEPNet Gate NU gating, §2.5), all trained under identical features and data.

Results. In Table 1, TGNN achieves an AUC of 83.24 and 85.29 on Datasets A and B, versus 59.57 and 58.81 for HiGNN, exceeding the strongest external baseline, HAN (Wang et al., 2019b), by $+18.55$ and $+13.87$ AUC points (relative gains of $+28.7\%$/$+19.4\%$). The external baselines

Table 1: Offline comparison under Dataset A and B scenarios: three external GNN baselines, the production baseline HiGNN and the full TGNN, over all ten metrics (R@K/P@K/F1@K = Recall/Precision/F1@K). External baselines are re-implemented on the same graph, features and training protocol as TGNN; complete ablation results are in Table 5 (Appendix C.3). In each column the best value is in **bold** and the runner-up is underlined.

Scen.	Framework	AUC	R@1	R@20	R@100	P@1	P@20	P@100	F1@1	F1@20	F1@100
A	SimGCL	63.40	0.0404	0.1545	0.2239	0.0485	0.0518	0.0574	0.0441	0.0776	0.0914
	GraphSAGE	64.09	0.0484	0.1266	0.2091	0.0549	0.0569	0.0570	0.0514	0.0786	0.0895
	HiGNN	59.57	0.0270	0.0982	0.1673	0.0568	0.0607	0.0573	0.0366	0.0750	0.0854
	HAN	64.69	0.0543	0.1701	0.2572	0.0582	0.0621	0.0619	0.0562	0.0910	0.0998
	TGNN	**83.24**	**0.1095**	**0.3011**	**0.4506**	**0.1082**	**0.1146**	**0.1105**	**0.1088**	**0.1661**	**0.1775**
B	SimGCL	70.50	0.0130	0.0825	0.1206	0.0625	0.0536	0.0550	0.0215	0.0650	0.0755
	GraphSAGE	70.95	0.0193	0.0806	0.1651	0.0736	0.0718	0.0678	0.0306	0.0760	0.0962
	HiGNN	58.81	0.0117	0.0599	0.1514	0.0486	0.0478	0.0565	0.0189	0.0532	0.0823
	HAN	71.42	0.0411	0.1329	0.2101	0.0872	0.0523	0.0773	0.0559	0.0751	0.1130
	TGNN	**85.29**	**0.0730**	**0.2418**	**0.4000**	**0.1689**	**0.1472**	**0.1256**	**0.1020**	**0.1830**	**0.1912**

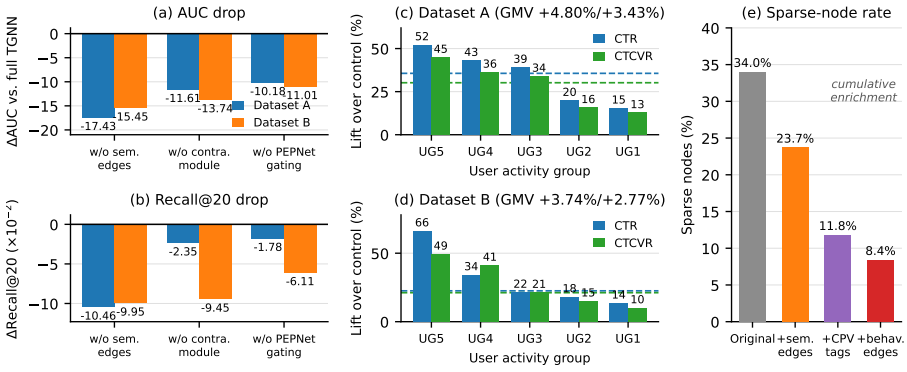

Figure 3: (a,b) Ablation study: performance drop of each ablated variant relative to the full TGNN (sem. = semantic edges (§2.3), contra. = hierarchical contrastive module (§2.4), PEPNet = gating (§2.5); complete ten-metric numbers in Table 5 (Appendix C.3)). (c,d) Online A/B results: *relative* CTR/CTCVR lifts in percent (not percentage points) over the control group per user-activity group (UG5 most active; same thresholds in both scenarios); dashed lines mark the all-traffic totals; GMV in the titles is platform-level (new ads/total). (e) Fraction of sparse nodes (degree < 5) after each cumulative enrichment step (+sem. edges = LLM-based semantic edges, +CPV = Category-Property-Value tags, +behav. = cross-domain behavioural edges).

maintain the same ordering (SimGCL $<$ GraphSAGE $<$ HAN) on both scenarios, confirming that heterogeneous message passing matters at industrial scale.

4.2 Ablation Study

As visualized in panels (a,b) of Figure 3, the removal of LLM-enriched semantic edges (TGNN-w/o-SE) triggers by far the most substantial decline on both metrics (-17.43/-15.45 AUC and -0.1046/-0.0995 Recall@20 on Datasets A/B), confirming that LLM-enhanced semantic knowledge greatly alleviates the data sparsity issue in industrial advertising; TGNN outperforms both TGNN-w/o-HC and TGNN-w/o-PG on every reported metric (Table 5). The other two components are complementary, as the mismatch between the two panels shows: on ranking-oriented metrics w/o-HC loses more AUC (-11.61/-13.74 vs. -10.18/-11.01), whereas on deep recall w/o-PG loses more (Recall@100 0.4205 vs. 0.4436 on Dataset A) — the contrastive module sharpens the ranking signal, PEPNet gating pulls additional relevant candidates into the shortlist.

4.3 Online A/B Test

Experimental setup. Online traffic is randomly partitioned into equal-sized buckets (control vs. treatment); each experiment runs for several weeks, and totals are computed over all traffic (not the unweighted mean of per-group lifts). The control is the deployed production retrieval system; TGNN is added as one retrieval branch under an identical quota, isolating the effect of the branch itself rather than a change in the retrieval budget.

Metrics and measurement scope. We measure Click-Through-Rate (CTR), Click-Through-and-Conversion Rate (CTCVR) and the relative growth in Gross Merchandise Volume (GMV), overall and for new advertisements, and partition users into five equally sized activity groups UG1–UG5, ordered from the least to the most active, using the same thresholds in both scenarios. Unless otherwise stated, CTR and CTCVR are *branch-level* (computed only over TGNN-branch impressions against the control branch under an identical quota), whereas GMV is *platform-level* (aggregated over all traffic regardless of the serving branch). The two are therefore not directly comparable: branch lifts reflect the quality of the branch itself, platform GMV its marginal contribution after competing with all other branches in ranking and auction; we regard the latter as the decisive evidence of business value.

Results. Panels (c,d) of Figure 3 report the per-group CTR/CTCVR lifts (relative, in percent): the TGNN branch delivers +35.62% CTR / +30.17% CTCVR on Dataset A and +22.53% / +21.17% on Dataset B. Gains scale with user activity: the most active group (UG5) reaches +52.21% and +66.32% CTR on the two datasets, while even UG1 shows a double-digit increase. The CTCVR lift is almost entirely inherited from CTR (CTCVR $=$ CTR$\times$CVR): the implied per-click conversion rate stays essentially flat (-4.02%/-1.11% on A/B), i.e. TGNN earns more clicks without diluting post-click conversion quality. TGNN increases total GMV by +3.43% and +2.77%, and by +4.80% and +3.74% for new advertisements, which is direct evidence that semantic enrichment and warm-start initialization overcome the cold-start barrier.

4.4 Influence on Data Sparsity

We define sparse nodes as those with a degree below five: the original graph is severely disconnected (34.0% of nodes), and enrichment alleviates this *cumulatively* (panel (e) of Figure 3): semantic edges bring the rate to 23.7%, CPV tags to 11.8%, cross-domain behavioural edges to 8.4% — the dense structural foundation for modeling cold-start entities and low-activity users.

4.5 Validation on public benchmark dataset

To validate generality beyond proprietary scenarios, we evaluate TGNN against SGL-ED (Wu et al., 2021), MixGCF (Huang et al., 2021), SimGCL (Yu et al., 2022) and XSimGCL (Yu et al., 2023) on Amazon Books (Ni et al., 2019) and Yelp-2018 (He et al., 2020) (Table 6, Appendix C.4). TGNN achieves the best Recall@20 on both datasets (0.0613 vs. 0.0502; 0.0818 vs. 0.0733); its NDCG@20 is best on Amazon Books (0.0457 vs. 0.0411) and competitive on Yelp-2018 (0.0601, marginally below XSimGCL's 0.0606).

5 Conclusion

We presented TGNN, which overcomes the key bottlenecks of deploying GNNs in industrial advertising: LLM-constructed semantic edges, cross-domain augmentation, hierarchical contrastive objectives and gating-based fusion address extreme sparsity, realized through a hardware–software co-design scaling to two billion nodes with minute-level freshness. Extensive A/B testing demonstrates its commercial value and that algorithmic design must be co-optimized with the underlying system.

AI use statement

In preparing this work, we used generative AI tools only for (i) copy-editing and language polishing of the manuscript text, and (ii) routine coding assistance in code outside the model, the training pipeline,

and the experimental evaluation. All research ideas, the method design, the system implementation, the experiments, and the analysis were conceived and carried out entirely by the authors; no generative AI tool was used to produce experimental numbers, figures, tables, or proofs. All AI-assisted text and code were reviewed, verified, and tested by the authors, who take full responsibility for the final content of this work.

Ethics Statement

This work studies retrieval models for online advertising. All data used in our experiments are behavioural interaction logs that are already collected, aggregated and de-identified by the serving platform under its existing privacy policy and internal compliance review; no personally identifiable information, and no text written by individual users, is used, stored or exposed in this paper. Industrial results are reported only as aggregate relative lifts over large user groups, which prevents any inference about individuals. The public-benchmark experiments use only the Amazon Books and Yelp-2018 datasets, which are publicly available research datasets. We do not conduct any study involving human subjects, and we do not release models or data that could be used to re-identify users. We acknowledge that improving ad retrieval accuracy can affect what content users see; in our deployment this is governed by the platform's existing creative-review, frequency-capping and diversity mechanisms, and we view careful measurement of such downstream effects as an important direction for future work.

Reproducibility Statement

We aim to make the reproducible core of this work as explicit as possible. The problem formulation, the model components (semantic-edge construction, multi-view contrastive objective, gating-based fusion) and the objective functions are specified in Section 2; the storage and training engines are described in Section 3 with their low-level designs deferred to Appendices B, D and E; the complete offline comparison including external GNN baselines is reported in Section 4 and Appendix C.3; and the full production hyperparameter configuration is listed in Appendix G. The public-benchmark experiments follow standard evaluation protocols on publicly available datasets (Amazon Books and Yelp-2018) with publicly described baselines, and are therefore externally reproducible. The industrial experiments cannot be reproduced externally because they depend on a proprietary serving platform and on user interaction logs that cannot be released; we nonetheless report the full metric definitions, the bucket-based evaluation protocol and the ablation structure so that the reported comparisons can be checked for internal consistency. We will release the code of the model and training pipeline upon publication, subject to internal review.

References

Anirudhan Badrinath, Alex Yang, Kousik Rajesh, Prabhat Agarwal, Jaewon Yang, Haoyu Chen, Jiajing Xu, and Charles Rosenberg. Omnisage: Large scale, multi-entity heterogeneous graph representation learning. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 4261–4272, 2025.

Oren Barkan and Noam Koenigstein. Item2vec: neural item embedding for collaborative filtering. In *2016 IEEE 26th international workshop on machine learning for signal processing (MLSP)*, pp. 1–6. IEEE, 2016.

Fedor Borisyuk, Shihai He, Yunbo Ouyang, Morteza Ramezani, Peng Du, Xiaochen Hou, Chengming Jiang, Nitin Pasumarthy, Priya Bannur, Birjodh Tiwana, et al. Lignn: Graph neural networks at linkedin. In *Proceedings of the 30th ACM SIGKDD conference on knowledge discovery and data mining*, pp. 4793–4803, 2024.

Jianxin Chang, Chenbin Zhang, Yiqun Hui, Dewei Leng, Yanan Niu, Yang Song, and Kun Gai. Pepnet: Parameter and embedding personalized network for infusing with personalized prior information. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 3795–3804, 2023.

Xu Cheng, Liang Yao, Feng He, Yukuo Cen, Yufei He, Chenhui Zhang, Wenzheng Feng, Hongyun Cai, and Jie Tang. Lps-gnn: Deploying graph neural networks on graphs with 100-billion edges. *ACM Transactions on Knowledge Discovery from Data*, 2025.

Alin Deutsch, Yu Xu, Mingxi Wu, and Victor Lee. Tigergraph: A native mpp graph database. *arXiv preprint arXiv:1901.08248*, 2019.

Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

Swapnil Gandhi and Anand Padmanabha Iyer. P3: Distributed deep graph learning at scale. In *15th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 21)*, pp. 551–568, 2021.

Wenwen Gong, Yangliao Geng, Dan Zhang, Yifan Zhu, Xiaolong Xu, Haolong Xiang, Amin Beheshti, Xuyun Zhang, and Lianyong Qi. Divgcl: A graph contrastive learning model for diverse recommendation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 16853–16861, 2025.

Joseph E Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. {PowerGraph}: Distributed {Graph-Parallel} computation on natural graphs. In *10th USENIX symposium on operating systems design and implementation (OSDI 12)*, pp. 17–30, 2012.

Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. Deepfm: a factorization-machine based neural network for ctr prediction. *arXiv preprint arXiv:1703.04247*, 2017.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pp. 639–648, 2020.

Tinglin Huang, Yuxiao Dong, Ming Ding, Zhen Yang, Wenzheng Feng, Xinyu Wang, and Jie Tang. Mixgcf: An improved training method for graph neural network-based recommender systems. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*, pp. 665–674, 2021.

Jiawei Jiang, Pin Xiao, Lele Yu, Xiaosen Li, Jiefeng Cheng, Xupeng Miao, Zhipeng Zhang, and Bin Cui. Psgraph: How tencent trains extremely large-scale graphs with spark? In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pp. 1549–1557. IEEE, 2020.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

Mahesh Lal. *Neo4j graph data modeling*. Packt Publishing Ltd, 2015.

Doyup Lee, Chiheon Kim, Saehoon Kim, Minsu Cho, and Wook-Shin Han. Autoregressive image generation using residual quantization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11523–11532, 2022.

Chaozhuo Li, Bochen Pang, Yuming Liu, Hao Sun, Zheng Liu, Xing Xie, Tianqi Yang, Yanling Cui, Liangjie Zhang, and Qi Zhang. Adsgnn: Behavior-graph augmented relevance modeling in sponsored search. In *Proceedings of the 44th international ACM SIGIR conference on research and development in information retrieval*, pp. 223–232, 2021.

Mu Li, Li Zhou, Zichao Yang, Aaron Li, Fei Xia, David G Andersen, and Alexander Smola. Parameter server for distributed machine learning. In *BigLearn Workshop, Advances in Neural Information Processing Systems*, 2013.

Zekun Li, Zeyu Cui, Shu Wu, Xiaoyu Zhang, and Liang Wang. Fi-gnn: Modeling feature interactions via graph neural networks for ctr prediction. In *Proceedings of the 28th ACM international conference on information and knowledge management*, pp. 539–548, 2019.

Tongtong Liu, Zhaohui Wang, Meiyue Qin, Zenghui Lu, Xudong Chen, Yuekui Yang, and Peng Shu. Real-time ad retrieval via llm-generative commercial intention for sponsored search advertising. *arXiv preprint arXiv:2504.01304*, 2025.

Jianmo Ni, Jiacheng Li, and Julian McAuley. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pp. 188–197, 2019.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

Feiyang Pan, Shuokai Li, Xiang Ao, Pingzhong Tang, and Qing He. Warm up cold-start advertisements: Improving ctr predictions via learning to learn id embeddings. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 695–704, 2019.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

Cheng Wan, Youjie Li, Ang Li, Nam Sung Kim, and Yingyan Lin. Bns-gcn: Efficient full-graph training of graph convolutional networks with partition-parallelism and random boundary node sampling. *Proceedings of Machine Learning and Systems*, 4:673–693, 2022.

Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, et al. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2019a.

Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. Heterogeneous graph attention network. In *The world wide web conference*, pp. 2022–2032, 2019b.

Jiancan Wu, Xiang Wang, Fuli Feng, Xiangnan He, Liang Chen, Jianxun Lian, and Xing Xie. Self-supervised graph learning for recommendation. In *Proceedings of the 44th international ACM SIGIR conference on research and development in information retrieval*, pp. 726–735, 2021.

Min Wu, Xinglu Yi, Hui Yu, Yu Liu, and Yujue Wang. Nebula graph: An open source distributed graph database. *arXiv preprint arXiv:2206.07278*, 2022.

Chang Xue, Youwei Lu, Chen Yang, and Jinming Xing. Recmind: Llm-enhanced graph neural networks for personalized consumer recommendations. *arXiv preprint arXiv:2509.06286*, 2025.

Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 974–983, 2018.

Junliang Yu, Hongzhi Yin, Xin Xia, Tong Chen, Lizhen Cui, and Quoc Viet Hung Nguyen. Are graph augmentations necessary? simple graph contrastive learning for recommendation. In *Proceedings of the 45th international ACM SIGIR conference on research and development in information retrieval*, pp. 1294–1303, 2022.

Junliang Yu, Xin Xia, Tong Chen, Lizhen Cui, Nguyen Quoc Viet Hung, and Hongzhi Yin. Xsimgcl: Towards extremely simple graph contrastive learning for recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 36(2):913–926, 2023.

Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In *2nd USENIX workshop on hot topics in cloud computing (HotCloud 10)*, 2010.

Dalong Zhang, Xin Huang, Ziqi Liu, Jun Zhou, Zhiyang Hu, Xianzheng Song, Zhibang Ge, Lin Wang, Zhiqiang Zhang, and Yuan Qi. Agl: a scalable system for industrial-purpose graph machine learning. *Proceedings of the VLDB Endowment*, 13(12):3125–3137, 2020.

Tong Zhao, Yozen Liu, Matthew Kolodner, Kyle Montemayor, Elham Ghazizadeh, Ankit Batra, Zihao Fan, Xiaobin Gao, Xuan Guo, Jiwen Ren, et al. Gigl: Large-scale graph neural networks at snapchat. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 5225–5236, 2025.

Shichao Zhu, Mufan Li, Guangmou Pan, and Xixun Lin. Ttgl: Large-scale multi-scenario universal graph learning at tiktok. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 5249–5259, 2025.

Yongchun Zhu, Ruobing Xie, Fuzhen Zhuang, Kaikai Ge, Ying Sun, Xu Zhang, Leyu Lin, and Juan Cao. Learning to warm up cold item embeddings for cold-start recommendation with meta scaling and shifting networks. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 1167–1176, 2021.

Yuanhang Zou, Zhihao Ding, Jieming Shi, Shuting Guo, Chunchen Su, and Yafei Zhang. Embedx: A versatile, efficient and scalable platform to embed both graphs and high-dimensional sparse data. *Proceedings of the VLDB Endowment*, 16(12):3543–3556, 2023.

A Appendix

This appendix is organized as follows. Appendix B overviews the AngelGraph architecture, consolidates the production deployment statistics (Appendix B.1) and compares TGNN with industrial GNN systems (Appendix B.2); Appendix C reports additional experimental results, namely the notation table (Appendix C.1), the semantic-edge construction procedure (Appendix C.2) and the complete offline comparison including external GNN baselines (Appendix C.3); Appendices D, E and F detail the storage engine, the load-balancing mechanisms and the feature-oriented optimizations; Appendix G lists the production hyperparameters; and Appendix H gives the training procedure and the proof of Proposition 2.1.

B AngelGraph Architecture Overview

AngelGraph adopts a hybrid Parameter Server (PS) (Li et al., 2013) + Spark (Zaharia et al., 2010) architecture (Jiang et al., 2020). PS in AngelGraph efficiently manages and stores high dimensional node features and model parameters, while Spark orchestrates data preprocessing and distributed PyTorch (Paszke et al., 2019) scheduling. Within the Spark Executors, AngelGraph invokes EasyGraph's subgraph sampling services and organizes data batches into PyTorch for optimized computation. AngelGraph supports over 20 GNN models across homogeneous, bipartite, and heterogeneous graph structures, with native support for custom meta-paths, multi-subgraph contrastive learning, and subgraph aggregation. This flexibility allows fine-grained personalized modeling and multi-scenario fusion across diverse advertising traffic streams. The load-balancing mechanisms that sustain this scale (load-aware partitioning and dynamic task scheduling) are detailed in Appendix E.

B.1 Production Deployment Statistics and Efficiency

Table 2 consolidates the production deployment statistics referenced throughout the paper. A naive prototype that stacked the GNN encoder on top of the existing retrieval stack incurred $5\times$ the serving latency of the base system, which is infeasible for production; the stream–batch architecture (Section 3), unified storage with $3.8\times$ feature compression and request coalescing, and the optimizations of Appendices D–F jointly reduce this to the sub-10 ms TP-999 profile below.

Table 2: Production deployment statistics of TGNN.

Item	Value
Graph scale	2×10^9 nodes, 10^{11} edges
LLM semantic I2I edges	1.2×10^9 (covering 92% of ads)
Daily incremental training samples	2×10^8
Daily inference requests	3.4×10^{10}
Subgraph sampling	2-hop (5×5), million-level sampling QPS
Sampling latency	TP-999 < 10 ms
Embedding freshness	measured < 2 min near-line refresh (SLA: 5 min)
Feature storage	$3.8\times$ compression, SWMR cache, MVCC versioning
Train–serve consistency	timestamp-versioned shared storage

B.2 Comparison with Industrial GNN Systems

Table 3 positions TGNN against representative industrial GNN systems along the axes raised in Section 1: scale, heterogeneity, storage design, load balancing and inference latency, as characterized in the respective publications.

Table 3: Qualitative comparison with representative industrial GNN systems, as characterized in their respective publications.

System	Scale	Hetero.	Storage	Inference & latency
PSGraph (Jiang et al., 2020)	10^9 nodes / 10^{10} edges	homo.	Spark + separated parameter server	batch inference only
EmbedX (Zou et al., 2023)	—	homo.	separated stores	$\sim$20 ms (homo. only)
LPS-GNN (Cheng et al., 2025)	—	homo.	—	single-GPU, homo. only
LiGNN (Borisyuk et al., 2024)	10^9 nodes/edges	homo.	separated stores	online; latency not reported
GiGL (Zhao et al., 2025)	10^9 nodes / 10^{10} edges	homo.	separated stores	batch + optional online
OmniSage (Badrinath et al., 2025)	10^9 nodes / 10^{10} edges	multi-entity	separated stores	daily batch
EasyGraph + AngelGraph (ours)	$\mathbf{2 \times 10^9}$ nodes / $\mathbf{10^{11}}$ edges	**homo.+hetero.**	**unified feature+graph (SWMR, MVCC)**	**stream–batch; TP-999 $<$10 ms; $<$2 min freshness**

C Additional Experimental Results

C.1 Notations

Table 4: Main notations used in Section 2.

Notation	Description
$\mathcal{G}$	Heterogeneous graph of the advertising ecosystem
$\mathcal{U}, \mathcal{A}, \mathcal{S}$	Sets of user, advertisement and semantic attribute nodes
$\mathcal{E}^{beh}, \mathcal{E}^{sem}$	Behavioral and semantic edge sets
ϕ, ψ	Node-type and edge-type mappings
w_{vu}	Edge strength, used as the logit bias $\log w_{vu}$
$\mathbf{h}_v^{(l)}$	Hidden representation of node v at layer l
$\mathbf{e}_v^{beh}, \mathbf{e}_v^{sem}$	Behavior-view and semantic-view embeddings
$\mathrm{sid}(a)$	Semantic ID $(c_a^1, \ldots, c_a^M)$ of ad a
$\ell(a, b)$	Common-prefix depth of two semantic IDs
$\mathcal{T}_{CPV}(a)$	CPV tag set of ad a
$\mathcal{L}_{main}, \mathcal{L}_{ssl}$	Main (sampled-softmax) and contrastive losses
$\pi_d(u), g_a^{beh}, g_a^{sem}$	User-side and ad-side gating weights

C.2 Semantic Edge Construction Procedure

Algorithm 1 summarizes the two-stage construction of $\mathcal{E}^{sem}$ described in Section 2.

Algorithm 1 Semantic edge construction (I2I via the semantic-ID trie; I2CPV via hierarchical tags).

Require: Ad attributes $\{x_a\}_{a\in\mathcal{A}}$, CPV tag sets $\{\mathcal{T}_{CPV}(a)\}$, interaction logs; depth M, codebook size K, threshold $\ell_{\min}$, degree caps D, D_t
Ensure: Semantic edge set $\mathcal{E}^{sem} = \mathcal{E}^{i2i} \cup \mathcal{E}^{cpv}$
▷ *Stage 1: semantic IDs and I2I edges*
$\mathbf{s}_a \leftarrow \mathrm{LLM}(x_a)$; learn codebooks $\{\mathbf{e}_k^l\}_{k,l}$ by the RQ-VAE objective ▷ straight-through
for $a \in \mathcal{A}$ **do**
  Build $\mathrm{sid}(a) = (c_a^1, \ldots, c_a^M)$ by Eq. 5; insert it into trie $\mathcal{T}$
end for
for $a \in \mathcal{A}$ **do**
  $\mathcal{N}^{i2i}(a) \leftarrow$ Top-D ads sharing a prefix of depth $\geq \ell_{\min}$ in $\mathcal{T}$, ranked by Eq. 6 ▷ $O(M)$ descent
end for
▷ *Stage 2: I2CPV edges via hierarchical tags*
for $u \in \mathcal{U}$ **do**
  $T_u \leftarrow$ Top-D_t recency-weighted tags of $\bigcup_{a\in\mathcal{A}_u^+} \mathcal{T}_{CPV}(a)$ ▷ Eq. 8
end for
return $\mathcal{E}^{sem} \leftarrow \{(a, b, w_{ab}^{i2i}) : b \in \mathcal{N}^{i2i}(a)\} \cup \{(a, t, w_{a,t}^{cpv}) : t \in \mathcal{T}_{CPV}(a)\} \cup \{(u, t, w_{u,t}^{cpv}) : t \in T_u\}$

C.3 Complete Offline Comparison

Table 5 reports the complete offline comparison (ten metrics $\times$ eight models $\times$ two scenarios), including the three TGNN ablations whose effect on the headline metrics is visualized in Figure 3 of the main text. Below we collect the observations that are useful when reading the table.

External baselines. SimGCL (Yu et al., 2022), GraphSAGE (Hamilton et al., 2017) and HAN (Wang et al., 2019b) are homogeneous or heterogeneous GNN baselines trained on exactly the same graph, features and next-day prediction protocol as TGNN, so the comparison isolates the architecture rather than the data pipeline. Their relative order is stable across both scenarios (SimGCL $<$ GraphSAGE $<$ HAN), which is consistent with the claim that heterogeneous message passing is what matters at industrial scale.

Cross-metric ordering. The production baseline HiGNN has the lowest AUC on both scenarios, yet it is *not* last on every precision metric: on Dataset A its Precision@1 (0.0568) exceeds SimGCL (0.0485) and GraphSAGE (0.0549), and on Dataset B its Precision@100 (0.0565) exceeds SimGCL

Table 5: Complete offline comparison under Dataset A and Dataset B scenarios, covering all ten metrics for three external GNN baselines, the production baseline HiGNN, three TGNN ablations and the full model. Column abbreviations follow Table 1; SE = semantic edges, HC = hierarchical contrastive module, PG = PEPNet gating. In each column the best value is in **bold** and the runner-up is underlined.

Scen.	Framework	AUC	R@1	R@20	R@100	P@1	P@20	P@100	F1@1	F1@20	F1@100
A	SimGCL	63.40	0.0404	0.1545	0.2239	0.0485	0.0518	0.0574	0.0441	0.0776	0.0914
	GraphSAGE	64.09	0.0484	0.1266	0.2091	0.0549	0.0569	0.0570	0.0514	0.0786	0.0895
	HiGNN	59.57	0.0270	0.0982	0.1673	0.0568	0.0607	0.0573	0.0366	0.0750	0.0854
	HAN	64.69	0.0543	0.1701	0.2572	0.0582	0.0621	0.0619	0.0562	0.0910	0.0998
	TGNN-w/o-SE	65.81	0.0683	0.1965	0.2872	0.0587	0.0642	0.0613	0.0631	0.0967	0.1011
	TGNN-w/o-HC	71.63	0.0882	0.2776	0.4436	0.0931	0.1098	0.1056	0.0906	0.1573	0.1706
	TGNN-w/o-PG	73.06	0.0996	0.2833	0.4205	0.0967	0.1085	0.1035	0.0981	0.1569	0.1662
	TGNN	**83.24**	**0.1095**	**0.3011**	**0.4506**	**0.1082**	**0.1146**	**0.1105**	**0.1088**	**0.1661**	**0.1775**
B	SimGCL	70.50	0.0130	0.0825	0.1206	0.0625	0.0536	0.0550	0.0215	0.0650	0.0755
	GraphSAGE	70.95	0.0193	0.0806	0.1651	0.0736	0.0718	0.0678	0.0306	0.0760	0.0962
	HiGNN	58.81	0.0117	0.0599	0.1514	0.0486	0.0478	0.0565	0.0189	0.0532	0.0823
	HAN	71.42	0.0411	0.1329	0.2101	0.0872	0.0523	0.0773	0.0559	0.0751	0.1130
	TGNN-w/o-SE	69.84	0.0483	0.1423	0.2368	0.0911	0.0781	0.0734	0.0631	0.1008	0.1148
	TGNN-w/o-HC	71.55	0.0486	0.1473	0.2508	0.1030	0.0811	0.0750	0.0660	0.1046	0.1155
	TGNN-w/o-PG	74.28	0.0545	0.1807	0.2757	0.1241	0.0966	0.0821	0.0757	0.1259	0.1265
	TGNN	**85.29**	**0.0730**	**0.2418**	**0.4000**	**0.1689**	**0.1472**	**0.1256**	**0.1020**	**0.1830**	**0.1912**

Table 6: Evaluation results on public benchmark datasets (best in **bold**, runner-up underlined).

Frameworks	Amazon Books		Yelp-2018	
	Recall@20	NDCG@20	Recall@20	NDCG@20
SGL-ED	0.0478	0.0379	0.0675	0.0555
MixGCF	0.0451	0.0341	0.0717	0.0584
SimGCL	0.0490	0.0398	0.0721	0.0601
XSimGCL	0.0502	0.0411	0.0733	**0.0606**
TGNN	**0.0613**	**0.0457**	**0.0818**	0.0601

(0.0550). This is expected rather than contradictory: the production model is calibrated for the head of the ranking, so it retains competitive precision at small K while losing on AUC and Recall, which are dominated by coverage of the full candidate pool.

Runner-up marking. TGNN is the best entry in every column of Table 5. The underlined runner-up switches between TGNN-w/o-PG and TGNN-w/o-HC depending on the metric, which is exactly the complementarity discussed in Section 4: on Dataset B the runner-up is w/o-PG everywhere, whereas on Dataset A it is w/o-PG for the shallow ranking metrics (R@1, R@20, P@1, F1@1) but w/o-HC for AUC, deep recall (R@100) and the large-K precision/F1 metrics. In other words, the contrastive module contributes most to the ranking signal (AUC) and PEPNet gating to deep recall (Recall@100), matching Figure 3.

C.4 Public Benchmark Results

D EasyGraph Design Details

EasyGraph consists of three primary components:

- Proxy Server: Facilitates request translation and intelligent task scheduling to balance the sampling load.
- Cache Server: Implements a high-speed caching layer for graph structures to accelerate frequently accessed subgraphs.
- KV Storage: Provides a persistent and scalable backend for massive graph data.

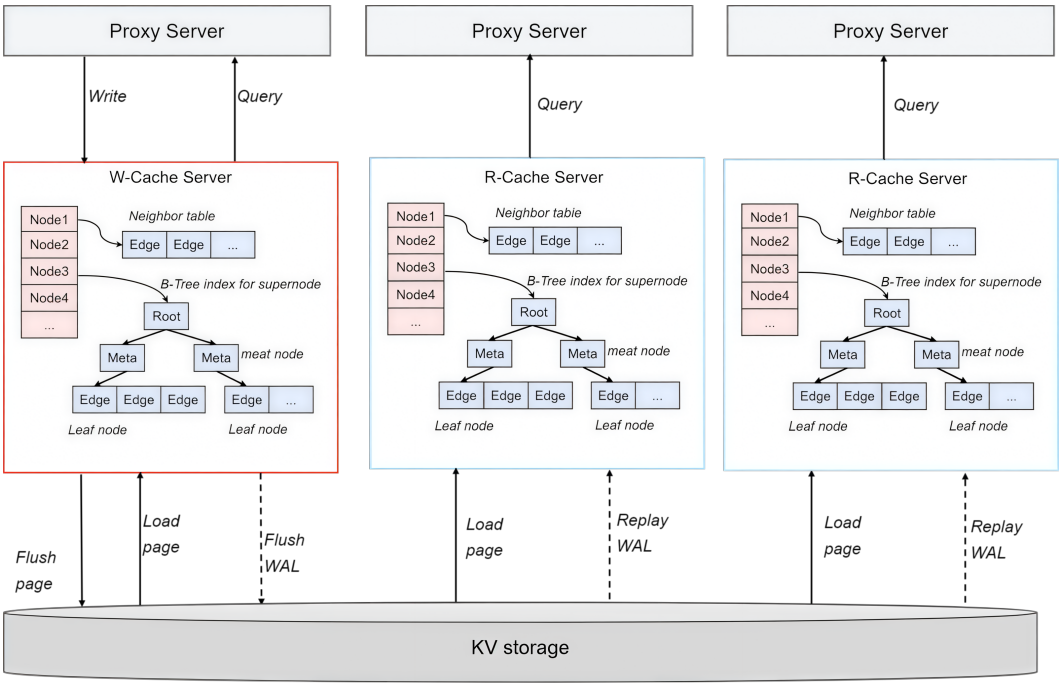

Figure 4: An overview of EasyGraph architecture.

Every component within EasyGraph is designed for horizontal scalability, enabling the system to effortlessly support industrial-scale graphs exceeding billions of entities. An overview of EasyGraph architecture is shown in Figure 4.

D.1 Graph Storage in EasyGraph

To meet the low-latency demands of real-time GNNs, we developed Cache Server, a distributed in-memory component for hot graph data. To overcome memory-bound sampling bottlenecks, EasyGraph employs a compact memory layout that aggregates all outgoing edges into contiguous adjacency lists. By aligning these structures with CPU cache line boundaries, we minimize cache misses and maximize sampling throughput.

A primary challenge in web-scale graphs is the super-node phenomenon, which triggers memory hotspots. EasyGraph mitigates this via a B-Tree based partitioning strategy for adjacency lists. This allows edges to be indexed by attributes (e.g., timestamps or weights), enabling efficient range queries and transforming linear scans into logarithmic searches. This structure ensures predictable latency for dense subgraphs while balancing the write amplification typical of high-degree nodes.

For high availability, the system adopts a Single-Writer-Multiple-Reader (SWMR) replication strategy. The primary W-Cache handles data ingestion and consistency, while the secondary R-Cache scales read throughput horizontally for concurrent sampling. To support multi-region deployment, W-Cache servers persist Write-Ahead Logs (WAL) to KV storage; R-Cache servers then replay these logs asynchronously to achieve eventual consistency and ensure resilience against localized failures.

D.2 Graph Sampling in EasyGraph

EasyGraph supports versatile interfaces for Random, Weighted, and Top-K sampling. For weighted sampling, the system flexibly selects between two strategies based on available resources and node degrees: Alias Table Sampling, which offers $O(1)$ complexity at the cost of double memory usage, and Binary Search Sampling, which saves 25% memory but exhibits $O(\log N)$ complexity.

Minimizing end-to-end latency is critical for online inference. EasyGraph implements a multi-tiered optimization strategy. Firstly, the RPC framework in EasyGraph maintains a dynamic map of replica

locations (region/IDC) and network latency, ensuring requests are always routed to the nearest available replica. Secondly, given that graph partitions far exceed physical instances, naive sampling would trigger excessive RPCs. EasyGraph mitigates this by coalescing requests destined for the same physical instance. This strategy effectively cuts the TP-999 latency by over 10%. Thirdly, EasyGraph compresses node features via Bitmap, Protobuf, and zstd. This compression strategy achieves a 3.8x compression ratio, reducing bandwidth and lowering TP-999 latency by another 10%. Overall, EasyGraph guarantees a TP-999 latency less than 10ms for standard 2-hop (5×5) sampling.

To ensure offline-online consistency, EasyGraph implements Multi-Version Concurrency Control (MVCC) based on time windows. By strictly restricting sampling to a specified timestamp, the system enables precise model backtracking and prevents temporal data leakage (future interaction bias).

E AngelGraph Design Details

E.1 Load-Aware Graph Partitioning

Standard industrial solutions (Gonzalez et al., 2012) typically use vertex-cuts for super nodes, but this incurs prohibitive cross-node communication overhead during GNN backward passes. To address this, TGNN introduces Load-Aware Graph Partitioning: *(1): Global Smart Edge-Cut*. We employ a greedy bucket assignment strategy based on adjacency size. Super nodes are isolated into dedicated partitions without being split across machines, while smaller nodes are packed into remaining capacity. This ensures workload equilibrium without cross-machine node splitting. *(2): Intra-Partition Vertex Slicing*. If a super node's adjacency list exceeds mini-batch capacity, we locally slice it into sub-segments. Because all segments reside on the same physical worker, aggregation requires only local memory access. This way, the communication overhead is eliminated. To further accelerate training, sampled subgraphs are encoded into continuous memory using a hybrid format: dense matrices for node features and CSR/CSC formats for topologies. This layout minimizes memory fragmentation and enables high-performance matrix operations during feature mapping and subgraph aggregation.

E.2 Dynamic Task Scheduling

While static partitioning provides a foundation for load balancing, it is insufficient for production-scale GNN training for two reasons: (1) static heuristics cannot perfectly predict the computational intensity of heterogeneous subgraphs, and (2) computation capability variation caused by soft resource isolation of the Hadoop cluster. To address this issue, AngelGraph adopts a *Dynamic Task Scheduling* approach. Concretely, we decompose the global training process into N fine-grained sub-tasks distributed among M compute units, where $N \gg M$. Instead of pre-assigning a fixed workload, AngelGraph utilizes a pull-based scheduling mechanism. Each compute node in AngelGraph fetches new sub-tasks as soon as they complete their current sub-tasks. This way, faster nodes autonomously claim a larger task volume, while slower "straggler" nodes handle fewer tasks. Dynamic task scheduling neutralizes the long-tail effect caused by data skew and hardware jitter, which greatly increases aggregate cluster throughput and reduces total training time.

F Additional Feature-Oriented Optimizations

F.1 Feature Initialization

AngelGraph supports heterogeneous feature types (e.g., dense, sparse, and sequential) and adopts dynamic initialization method for different features. AngelGraph adaptively learns the statistical characteristics of individual features and adaptively initialize features in a distribution-aware manner.

F.2 Feature Cold-start Strategy

For cold-start nodes (e.g., new advertisements), we leverage LLMs to compute semantic similarities between entities and bridge missing structural links with high-confidence semantic edges. We initialize sparse features for cold-start nodes by propagating historical embeddings from semantically similar "warm" neighbors instead of random initialization (Guo et al., 2017). This "warm-up" mechanism provides a sophisticated prior that significantly speeds up model convergence. Compared

to standard random initialization or heavy pre-training paradigms (Barkan & Koenigstein, 2016; Pan et al., 2019; Zhu et al., 2021), incorporating distribution priors and semantic knowledge help our model achieve superior performance in fewer training epochs. Further, our approach is computationally efficient and maintains distribution-level interpretability.

G HYPERPARAMETER SETTINGS

Table 7 reports the production hyperparameters of TGNN. Ad titles and descriptions are encoded with an open-source LLM encoder; the RQ-VAE semantic IDs of Section 2 are derived from these encodings, and I2I edges connect ads that share a trie prefix, which approximates cosine-similarity retrieval over the LLM embeddings. Training proceeds in an incremental stream–batch manner with the same configuration for offline evaluation and online serving; the contrastive- and RQ-VAE-related hyperparameters (λ_{ssl}, μ_1, μ_2, ε, M, K, $\ell_{\min}$ and D in Section 2) are tuned on a held-out split of each training window.

Table 7: Hyperparameter settings of TGNN in production.

Graph schema	
Node types	user, ad, product (CPV)
Feature fields (user / ad / product)	125 / 41 / 1
Per-field embedding dim (user / ad / product)	32 / 16 / 8
Field encoding	one-hot
Edge families	user–ad (6 relation types), ad–product, ad–ad
Meta-paths	7 (up to 4 hops)
Architecture	
Hidden dimension	64
Output embedding dimension	48
Negatives per positive	8
Multi-view fusion	co-attention
Sequence encoder	MLP-Mixer (1 layer), frozen
Time encoding	fixed-frequency cosine
Sequence length (user / ad / product)	5 / 10 / 5
Behavior sequence window	90 days
Topological mask threshold	0.5
Training	
Optimizer	Adam
Learning rate (main / sequence branch)	10^{-4} / 10^{-4}
Weight decay (main / sequence branch)	0 / 10^{-3}
Batch size	128
Epochs per incremental round	2
Neighbor fan-out	5 / 5 (weighted sampling; 2-hop)
Sample time window	7 days
Feature expiry	7 days
Serving	
Sampling feature batch (user / ad / product)	100 / 300 / 12500
Sampling workers	20
Retries / retry interval	3 / 50 ms
Backup-request timeout	100 ms
Connections per address	10

H TRAINING PROCEDURE AND PROOFS

Algorithm 2 One TGNN training step.

Require: Graph $\mathcal{G} = \mathcal{G}^{beh} \cup \mathcal{G}^{sem}$, mini-batch $\mathcal{B}$, encoder f_θ, gates of Eqs. 13–14

1: Sample subgraphs $(\mathcal{V}_\mathcal{B}, \mathcal{E}_\mathcal{B})$ around $\mathcal{B}$ ▷ heterogeneous sampling, Section 3
2: **for** $k \in \{1, 2\}$ **do**
3: Draw $\Delta_i^{(k)}$ by Eq. 9; $\mathbf{h}_i^{beh,(k)}, \mathbf{h}_i^{sem,(k)} \leftarrow f_\theta(\mathbf{x}_i + \Delta_i^{(k)})$ ▷ Eqs. 1–2
4: **end for**
5: $\mathbf{e}_u \leftarrow \sum_d \pi_d(u) O_{ep}^{(d)}$; $\mathbf{e}_a \leftarrow g_a^{beh}\mathbf{e}_a^{beh} + g_a^{sem}\mathbf{e}_a^{sem}$ ▷ Eqs. 13, 15
6: $\mathcal{L}_{ssl} \leftarrow \mathcal{L}_{intra} + \mu_1\mathcal{L}_{cross} + \mu_2\mathcal{L}_{top}$ ▷ Eq. 10
7: $\theta \leftarrow \theta - \eta\nabla_\theta\big(\mathcal{L}_{main} + \lambda_{ssl}\mathcal{L}_{ssl} + \lambda_{reg}\|\theta\|_2^2\big)$ ▷ Eq. 3

Proof of Proposition 2.1. At layer l the residual obeys $\|\mathbf{r}_a^l\|_2 \le \rho^l\|\mathbf{s}_a\|_2$, hence $\|\mathbf{e}_{c_a^l}^l\|_2 = \|\mathbf{r}_a^l - \mathbf{r}_a^{l+1}\|_2 \le \|\mathbf{r}_a^l\|_2 + \|\mathbf{r}_a^{l+1}\|_2 \le \rho^l(1+\rho)\|\mathbf{s}_a\|_2$. Since $\mathbf{z}_a = \sum_{l=1}^M \mathbf{e}_{c_a^l}^l$ and the two ads share their first ℓ codewords,

$$\|\mathbf{z}_a - \mathbf{z}_b\|_2 \le \sum_{l=\ell+1}^{M} \big(\|\mathbf{e}_{c_a^l}^l\|_2 + \|\mathbf{e}_{c_b^l}^l\|_2\big) \le 2(1+\rho)\|\mathbf{s}\|_2 \sum_{l=\ell+1}^{M} \rho^l \le 2(1+\rho)\|\mathbf{s}\|_2 \frac{\rho^{\ell+1}}{1-\rho}. \quad (16)$$

The quantization error at depth M is $\|\mathbf{s}_a - \mathbf{z}_a\|_2 = \|\mathbf{r}_a^{M+1}\|_2 \le \rho^M\|\mathbf{s}\|_2$, so the triangle inequality gives $\|\mathbf{s}_a - \mathbf{s}_b\|_2 \le 2\|\mathbf{s}\|_2\big((1+\rho)\frac{\rho^{\ell+1}}{1-\rho} + \rho^M\big)$. Finally $\cos(\mathbf{s}_a, \mathbf{s}_b) = 1 - \frac{\|\mathbf{s}_a - \mathbf{s}_b\|_2^2}{2\|\mathbf{s}\|_2^2}$, which yields the claim. □